Exploring the impact of low-rank adaptation on the performance, efficiency, and regularization of RLHF

Simeng Sun Dhawal Gupta Mohit Iyyer
University of Massachusetts Amherst
{simengsun,dgupta,miyyer}@cs.umass.edu

Abstract

During the last stage of RLHF, a large language model is aligned to human intents via PPO training, a process that generally requires large-scale computational resources. In this technical report, we empirically investigate an efficient implementation of RLHF using low-rank adaptation (LoRA), which allows us to align the LLaMA 7B checkpoint on the Alpaca dataset (Taori et al., 2023) using only two A100 GPUs instead of the eight required for full model fine-tuning. Despite tuning only 0.2% of LLaMA 7B's parameters, our implementation achieves *better* performance than the publicly-released AlpacaFarm checkpoint (Dubois et al., 2023) with full model fine-tuning. Next, we analyze several configurations of our LoRA-based PPO implementation, varying the form of the KL regularization term in the training objective. We find that (1) removing this penalty term does not harm performance on the AlpacaFarm evaluation set under our LoRA setup; (2) other regularizers, such as Jensen-Shannon divergence, lead to improved performance; and (3) while PPO training negatively impacts the factuality of model-generated responses, training with LoRA largely mitigates this effect. We release our code and pretrained checkpoints to facilitate future research on more efficient RLHF.[1]

1 Introduction

Reinforcement learning from human feedback (Ouyang et al., 2022, RLHF) is a technique used to align large language models (LLMs) with the intents of human users. While companies such as OpenAI, Google, and Anthropic provide blackbox access to LLMs tuned with RLHF, their codebases remain proprietary. While the development of open-source libraries such as TRL (von Werra et al., 2020), TRLX (Castricato et al., 2023), and AlpacaFarm (Dubois et al., 2023) has enabled other researchers to experiment with RLHF, the required experimental settings are computationally infeasible for most under-resourced labs. For example, aligning the pre-trained LLaMA 7B checkpoint (Touvron et al., 2023) with RLHF requires *eight* Nvidia A100 80GB GPUs using the AlpacaFarm library. Currently, a single node with 8 A100 GPUs costs around $200K to purchase,[2] and renting such resources from cloud providers borders on the impossible due to limited supply and huge demand.

Why does RLHF require so many GPUs? RLHF involves three stages that each requires fine-tuning a pre-trained LLM: (1) supervised fine-tuning on instruction following data; (2) reward model training on human preference data; and (3) fine-tuning via proximal policy optimization (Schulman et al., 2017b, PPO). The third stage, in which the RLHF-aligned LLM is actually created, is also the most expensive because it requires storing multiple large models (e.g., policy, value, reward, reference policy) along with gradients and optimizer states in GPU memory. The AlpacaFarm library leverages fully sharded data parallel training (Zhao et al., 2023, FSDP) to distribute the parameters

[1] `https://github.com/SimengSun/alpaca_farm_lora`
[2] For example, see this quote from Lambda Labs.

of these models across multiple GPUs, which along with flash attention (Dao et al., 2022) enables processing two examples per GPU.

Saving memory with LoRA: Low-rank adaptation (Hu et al., 2021, LoRA) is a parameter-efficient method to fine-tune large language models. In LoRA, the pre-trained LLM is frozen while only low-rank decomposition of the weight matrices (commonly just the projection matrices in self-attention) are optimized. The rank of the decomposition matrices is typically very small (e.g., 8 or 64) compared to the dimensionality of the hidden states (e.g., 4096), which greatly reduces memory consumption.

Contribution #1: In this technical report, we implement the third and most resource-demanding stage of RLHF (PPO) with LoRA and manage to bring down the hardware requirements from eight to two A100s. We observe no performance degradation when using LoRA; in fact, with just 10 hours of LoRA-based PPO training, we outperform the publicly-released AlpacaFarm checkpoint (trained via full model fine-tuning) in terms of win rate against `text-davinci-003`. To facilitate reproducibility and future research, we release our code, which modifies the AlpacaFarm library to support LoRA and alternative regularization schemes, as well as our LoRA weights for LLaMA-7B. All experiments in this report were conducted on two A100 80GB GPUs.

KL regularization may not be critical when using LoRA. The third stage of RLHF (i.e., PPO optimization) attempts to maximize the expected reward of the policy model while also penalizing large deviations from the pre-trained base model. This penalty is implemented via an approximation of KL divergence (Kullback and Leibler, 1951) between the policy and pre-trained reference LLM.

Contribution #2: While prior work has considered this penalty critical to the successful application of RLHF (Ouyang et al., 2022), our experiments show that the KL penalty can be completely removed when using LoRA without lowering the resulting model's win rate. Additionally, we discover that implementing this penalty using different divergence estimators (e.g., Jensen-Shannon divergence) can lead to higher win rates on the AlpacaFarm evaluation set. We hypothesize that LoRA itself acts as a powerful regularizer since most of the pretrained LLM's parameters (e.g., in the feed-forward layers) are left unchanged, and as such additional regularization is not as critical as in full model fine-tuning. This can lead to further memory reductions, as it may not be necessary to keep the reference policy in memory when using LoRA.

Limitations of this work: Due to limited compute (i.e., no access to a node with 8 A100 80GB GPUs), we could not run RLHF on the LLaMA 7B checkpoint using full model fine-tuning. Thus, it is infeasible for us to assess the impact of removing or replacing the KL penalty term on full model fine-tuning. These experiments are critical to verify our hypothesis about LoRA's regularization effect. Additionally, because FSDP implementations did not support disabling gradients for specific parameters at the time of our experiments, we do not use any model parallelism during training, which makes it difficult to experiment with the larger LLaMA checkpoints. In our current codebase, the policy and value models are stored on one GPU while the reference policy and reward model are stored on the other; the latter GPU is thus only sparsely utilized. Finally, as we only experiment with the AlpacaFarm data, our conclusions may not generalize to other domains, languages, or to more complex instruction sets.

2 Implementing RLHF with LoRA

To ground our discussion, we first provide a brief overview of the experimental settings that we consider in our work, which builds on the AlpacaFarm platform (Dubois et al., 2023). Then, we discuss how we incorporate LoRA into the PPO step of RLHF, which leads to slightly *improved* performance while also cutting down on memory consumption (from eight to two A100s).

2.1 Experimental settings

All of our experiments are conducted on the publicly-available LLaMA 7B checkpoint, which was pre-trained on 1 trillion tokens using 82K GPU hours (A100 80GB) (Touvron et al., 2023). We build our codebase on top of the open-source AlpacaFarm platform (Dubois et al., 2023) instead of other open-source RLHF libraries because AlpacaFarm also implements rigorous evaluations of RLHF-tuned instruction-following models.

RLHF data: We follow Dubois et al. (2023) by applying RLHF to the pre-trained LLaMA model using the Alpaca-52K instruction-following dataset (Taori et al., 2023). Each example in this dataset, which was automatically generated using `text-davinci-003` , contains an instruction for some task and a corresponding demonstration (i.e., an optional input and output for that task). The dataset contains a diverse set of tasks spanning both open-ended instructions (e.g., *Describe the impact of the coronavirus on the economy*) and highly constrained ones (*Standardize the following date to the ISO 8601 format*). Each of the three stages of RLHF is performed using a different split of the Alpaca-52K data: supervised fine-tuning with 10K examples, reward modeling with 10K instructions / model-generated outputs for which human raters provided preference judgments, and PPO with 20K instructions.[3]

Evaluation by win rate: AlpacaFarm collates existing open-source instruction-following datasets (Wang et al., 2023b; Bai et al., 2022a; Chiang et al., 2023; Geng et al., 2023) to form an evaluation dataset of 805 diverse instructions. To evaluate two models against each other on this dataset, we first collect responses generated by both models for all 805 instructions. Then, a pool of large language models (`GPT-4-0314`, `GPT-3.5-turbo`, `text-davinci-003`) are prompted to provide preference judgments (i.e., which model's output is better for a given instruction) by simulating human annotators, which allows us to compute the *win rate* of one model over the other. The LLM-based simulated workflows were shown to highly correlate with human raters in system-level comparison with a Spearman correlation of 0.98 Dubois et al. (2023). As in the original codebase, we compute the win rate of each of our model configurations against OpenAI's `text-davinci-003`.

Baseline model: Dubois et al. (2023) release checkpoints for the SFT-10K step (stage 1 of RLHF), the reward model trained on human preference (stage 2), and the PPO-optimized stage 3 checkpoint. Their SFT-10K model[4] reaches a win rate of 37% against `text-davinci-003`, while the PPO-optimized checkpoint[5] has a win rate of 47%.

In this technical report, we use the publicly-released AlpacaFarm checkpoints for the SFT and reward modeling stages of RLHF. These first two stages are not as memory intensive as the third stage,[6] and prior work (Hu et al., 2021; Santacroce et al., 2023) has shown that LoRA is very effective for fine-tuning LLMs with limited resources. Our focus is on the third stage of RLHF (PPO), in which these previous two checkpoints have to be kept in memory along with the policy network, where we perform PPO with LoRA using the third split of 20K instructions.

2.2 Reducing RLHF's memory consumption with LoRA

To alleviate the memory consumption of the third stage of RLHF, we employ low-rank adaptation (Hu et al., 2021) to align the model. In LoRA, an input hidden state $\mathbf{h}_{\text{in}} \in \mathbb{R}^d$ is projected to $\mathbf{h}_{\text{out}} \in \mathbb{R}^d$ via a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$ and two low-rank decomposition matrices $\mathbf{A} \in \mathbb{R}^{k\times d}$ and $\mathbf{B} \in \mathbb{R}^{d\times k}$, where k is the rank of the low-rank matrices ($k \ll d$) and α is a scaling hyperparameter:

$$\mathbf{h}_{\text{out}} = (\mathbf{W} + \frac{\alpha}{k}\mathbf{B}\mathbf{A})\mathbf{h}_{\text{in}} \tag{1}$$

During LoRA training, $\mathbf{W}$ is kept frozen while the decomposition matrices $\mathbf{A}$ and $\mathbf{B}$ are trained.

Experiment details: We follow the original LoRA setup of Hu et al. (2021) by setting the rank k to 8. Additionally, we set the scaling hyperparameter α to 64, which is a critical decision: using the default value of $\alpha = 1$ in the LoRA codebase[7] reduces the win rate by $\sim$ 6 points compared to when $\alpha = 64$. We apply LoRA only to the projection matrices (key, query, value, and output) in the attention layers. We also apply normal fine-tuning to the final projection head of size $[4096, 1]$ in the value function (critic), which leads to faster convergence in our experiments. Finally, we add dropout to all tuned layers with $p = 0.1$. In total, we optimize 16.7M parameters with LoRA ($\sim 0.2\%$ of

[3] The rest of the data is left unused.

[4] `https://huggingface.co/tatsu-lab/alpaca-farm-sft10k-wdiff`

[5] `https://huggingface.co/tatsu-lab/alpaca-farm-ppo-human-wdiff`

[6] Full-model fine-tuning for the first two stages requires 4 A100 80GB GPUs; however, only one A100 is needed if using LoRA.

[7] `https://github.com/microsoft/LoRA/tree/main`

Adaptation method	Regularization	Divergence estimator	Win rate
Publicly-released supervised fine-tuning (SFT-10k) checkpoint from Dubois et al. (2023)			
Full model tuning	-	-	37.0
Publicly-released PPO checkpoint from Dubois et al. (2023)			
Full model tuning	Clamped KL	$\max(0, \log \pi_\theta(y) - \log \pi_{\text{ref}}(y))$	46.7
Our PPO models trained with LoRA			
LoRA	Clamped KL	$\max(0, \log \pi_\theta(y) - \log \pi_{\text{ref}}(y))$	47.5 ± 0.2
LoRA	KL	$\log \pi_\theta(y) - \log \pi_{\text{ref}}(y)$	46.7 ± 0.02
LoRA	Bregman	$\frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)} - 1 - \log \frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)}$	49.0 ± 0.1
LoRA	Squared error	$\frac{1}{2}(\log \pi_\theta(y) - \log \pi_{\text{ref}}(y))^2$	47.1 ± 0.1
LoRA	Jensen-Shannon	$\frac{1}{2}\max(0, \log \pi_\theta(y) - \log \frac{1}{2}(\pi_\theta(y) + \pi_{\text{ref}}(y)))+$ $\frac{1}{2}\max(0, \log \pi_{\text{ref}}(y) - \log \frac{1}{2}(\pi_\theta(y) + \pi_{\text{ref}}(y)))$	49.8 ± 0.3
LoRA	*No regularization*	-	48.2 ± 0.1
LoRA	Dropout only (p=0.5)	-	48.4 ± 0.2

Table 1: Our experiments demonstrate that (1) LoRA is a competitive parameter adaptation method to full-model fine-tuning for PPO training in RLHF; and (2) PPO training with LoRA does not require any additional regularization (KL or otherwise) to succeed. We evaluate the win rate of each model in the table against `text-davinci-003` on the AlpacaFarm evaluation data. Preference judgments to calculate win rate are simulated by a pool of automated LLM annotators (e.g., GPT-4). We evaluate the public AlpacaFarm checkpoints in the top two rows (SFT-10k and PPO); for each of the remaining rows, we perform three runs of PPO optimization with LoRA and the corresponding regularizer, and we report the mean and standard error with bootstrap sampling.

LLaMA 7B's parameters). For all experiments, we set the KL coefficient $\beta = 0.02$ (see Eq. 3), the rollout batch size to 256, and the gradient update batch size to 128. We perform three training runs for each of our configurations and report mean and standard error with bootstrap sampling.

2.3 LoRA is effective for PPO training

Our experiments in Table 1 show that LoRA is a powerful parameter-efficient adaptation method for PPO training. While the publicly-released PPO-optimized checkpoint of Dubois et al. (2023) reaches a win rate of 46.7% with full model fine-tuning,[8] our corresponding LoRA model (third row of Table 1) obtains a slightly *higher* win rate of 47.5% despite optimizing just a small fraction of the parameters. Note that this is as controlled of an experiment as we can perform: we use the same KL regularization penalty as that of the published checkpoint and also the same batch size (16). We report win rate after 100 steps of training,[9] which takes around 10 hours of training on two A100s. The released AlpacaFarm PPO checkpoint was trained for 20 PPO steps,[10] which is equivalent to 40 steps in our experimental setup. Dubois et al. (2023) noted that optimal PPO performance is achieved between 20 to 80 PPO training steps under their setup, and they perform model selection based on simulated win rates.[11] In contrast, we observe increases in win rate until $\sim$100 steps in our LoRA setup (Figure 1). Having established the effectiveness of LoRA for RLHF, we now turn to investigating the regularization effects that it has on PPO training.

[8] We download their checkpoint and rerun the evaluation ourselves using pooled LLM annotators. While the AlpacaFarm leaderboard was recently updated to report a higher win rate of 49.25, our evaluation result closely matches their previously-reported number of 46.6 when using the pooled annotators before the update on Jun 23rd, 2023. (see here: `https://github.com/tatsu-lab/alpaca_farm/blob/0cfd0bce0506b3d68998c82b17a160d7da1d99a0/src/alpaca_farm/auto_annotations/eval.py#L23`).

[9] We report results after 100 steps because we empirically observe the win rate of most configurations plateaus after this point (see Figure 1 for win rates at other points during training).

[10] Verified after email communication with one of the authors.

[11] `https://github.com/tatsu-lab/alpaca_farm/issues/60`

Figure 1: Win rate against `text-davinci-003` on the AlpacaFarm evaluation set plotted as a function of PPO steps. The Jensen-Shannon estimator consistently outperforms no regularization, which in turn outperforms the standard KL estimator. We observe win rates plateauing after roughly 100 steps.

3 KL regularization in PPO training

First, we provide a brief overview of PPO training in the third stage of RLHF, highlighting the KL regularization term that is commonly regarded as critical for successful training. We experiment with different variants of this KL penalty with our LoRA setup (including no regularization at all), which we fully specify here.

3.1 PPO training

In the third stage of RLHF, the SFT model from the first stage (which is a model with basic instruction-following capabilities) is reinforced with PPO against the reward model trained in the second stage. PPO training (Schulman et al., 2017b) iterates between a rollout phase and an optimization phase.

Rollout: In rollout, the policy π_θ being optimized (i.e., the LM undergoing alignment) generates responses to a batch of input instructions. Each response is then assigned a scalar score by the reward model. This score is then used to estimate the advantage[12] for optimization in the next phase.

Optimization: The policy is optimized to maximize a surrogate objective using the advantage estimated during rollout. Then, the newly-optimized policy is used to generate responses in the next rollout phase. The surrogate objective in the optimization phase has the form

$$\mathcal{J}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}})], \tag{2}$$

where $r(\theta) = \frac{\pi_\theta(y|x)}{\pi_{\theta_{\text{old}}}(y|x)}$ denotes the output probability ratio between the current policy π_θ and the policy π_{old} from the previous rollout step, computed on responses y sampled during the previous rollout. $\hat{A}$ denotes the estimated advantage (Schulman et al., 2018), which depends on the reward value; we refer readers to other materials (Schulman et al., 2018; Weng, 2018) for understanding advantage estimation. The clipped output probability ratio stabilizes training by preventing large policy updates, thus discouraging π_θ from deviating too much from π_{old} during optimization.

3.2 KL Regularization

While the clipped ratio in Equation 2 constrains the extent to which π_θ can change from a recent policy π_{old}, π_θ can still reach a sub-optimal region by "reward hacking" (Pan et al., 2022) after enough rollout-optimization steps. To avoid this, a KL regularization term is added to penalize π_θ when it deviates too far from a reference policy π_{ref} during PPO training; the reference policy is usually set to the output of the first stage of RLHF (i.e., the SFT-10k checkpoint).

[12] A positive advantage suggests that when π_θ generates token y given a certain prefix, it receives a reward higher than the average reward expected from generating other tokens in the vocabulary given the same prefix.

Formally, let $\mathbf{x}$ be an instruction and $\mathbf{y}$ be a corresponding response of L tokens sampled during rollout. While the reward model produces a scalar value $r(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$ given the pair $(\mathbf{x}, \mathbf{y})$, the total reward $\mathbf{r}(\mathbf{x}, \mathbf{y})$ of the response $\mathbf{y}$ is a vector of dimensionality $\mathbb{R}^L$ due to the KL penalty:

$$\mathbf{r}(\mathbf{x}, \mathbf{y}) = [0 \quad 0 \quad \dots \quad r(\mathbf{x}, \mathbf{y})]^\top - \beta \mathrm{KL}(\pi_\theta(\mathbf{y} \mid \mathbf{x}), \pi_{\text{ref}}(\mathbf{y} \mid \mathbf{x})) \tag{3}$$

Here, $\mathrm{KL}(\cdot, \cdot)$ denotes the KL divergence (Kullback and Leibler, 1951) between π_θ and π_{ref} at each position (token) of $\mathbf{y}$, and the scalar reward r is added to the last position of the KL term. The shaped reward $\mathbf{r}$ is then used to estimate the advantage, which is at the core of PPO.

As it is memory-consuming to compute the KL divergence over the entire vocabulary at each timestep,[13] prior work approximates the KL term via Monte-Carlo estimation (Schulman et al., 2017a). Does the form of this approximation make an impact when implementing RLHF with LoRA? In this report, we compare the following divergence estimators based on win rate:

- **KL divergence:** Given the distributions over the vocabulary predicted by both the current policy π_θ and the reference policy π_{ref} at token y of the response, the Kullback-Leibler divergence is defined as $\mathrm{KL}(\pi_\theta(y), \pi_{\text{ref}}(y)) = \sum_{y \in \mathcal{V}} \pi_\theta(y) \log \frac{\pi_\theta(y)}{\pi_{\text{ref}}(y)}$. By minimizing the KL divergence, we effectively add an entropy regularizer that diversifies the generated tokens while also maintaining high likelihood under the reference policy. In practice (and in many open-source implementations), the true KL divergence is approximated as $\log \pi_\theta(y) - \log \pi_{\text{ref}}(y)$, which saves memory as only the probability of the generated token y is stored instead of the entire distribution. While this estimator is unbiased, it also suffers from high variance (Schulman, 2020); to enforce non-negativity, the AlpacaFarm implementation clamps the minimum value to zero, which reduces variance at the cost of biasing the estimator.
- **Alternative divergence approximations:** Schulman (2020) proposes two alternative KL approximations that have lower variance. The first takes the form of squared error between $\log \pi_\theta(y)$ and $\log \pi_{\text{ref}}(y)$, which is biased and approximates a different f-divergence measure.[14] The second one is the Bregman divergence $B_{\mathcal{F}}(\frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)}, 1)$ associated with the convex function $\mathcal{F}(x) := -\log x$, which is an unbiased KL estimator.[15] Both alternatives are shown to reduce variance and bias in a toy setting by Schulman (2020); in this report, we verify their effectiveness on real natural language tasks.
- **Jensen-Shannon divergence:** We also experiment with the Jensen-Shannon divergence, which is the average of the KL divergence for both π_{ref} and π_θ against the intermediate distribution $\pi_m(y) = \frac{1}{2}(\pi_{\text{ref}}(y) + \pi_\theta(y))$. Note that the responses are sampled from π_θ instead of π_{ref}. Similar to the AlpacaFarm implementation, the minimum value is clamped to zero for stabilized training. We fully specify the estimator in Table 1.
- **No KL regularization:** Finally, to evaluate whether KL regularization is necessary under the LoRA setup, we also discard KL penalty entirely in two of our configurations, one with dropout (Srivastava et al., 2014) and the other without.

4 Results & analysis

We observe several interesting phenomena while varying the regularization estimator within our LoRA RLHF setup. First and perhaps most notably, completely *removing* the KL regularization penalty does not affect win rate (it actually increases from 47.5% to 48.2% given same amount of

[13] Computing the true KL requires summing over the entire vocabulary, which is not economical if done for every token in the sequence.

[14] $f(x) := \frac{1}{2}(\log x)^2$

[15] It is an unbiased estimator of KL because

$$\begin{aligned}\mathbb{E}_{y \sim \pi_\theta}[\frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)} - 1 - \log \frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)}] &= -1 + \sum_{y \in V} \pi_\theta(y) \frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)} - \sum_{y \in V} \pi_\theta(y) \log \frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)} \\ &= -1 + 1 + \mathrm{KL}(\pi_\theta(y), \pi_{\text{ref}}(y))\end{aligned}$$

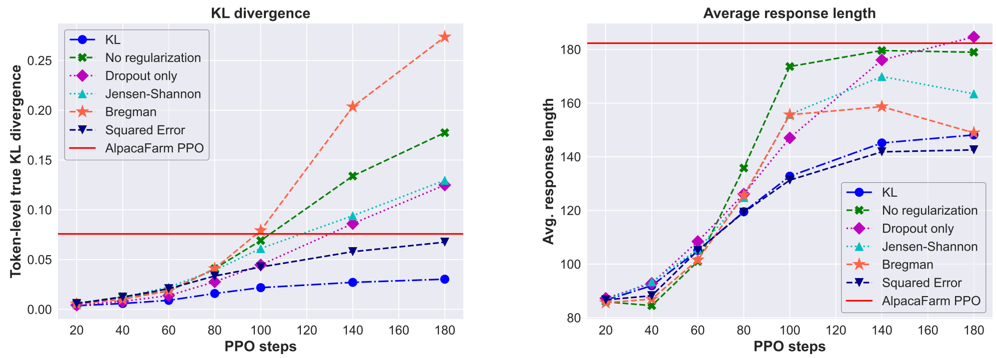

Figure 2: (**Left**): Regardless of the estimator, the true KL divergence (measured on 28K tokens sampled from AlpacaFarm) steadily increases as training proceeds. The standard KL estimator is predictably the most effective regularizer in terms of reducing true KL divergence. (**Right**): Response length increases until about 100 PPO steps, after which it plateaus or drops for most configurations.

compute). Of the alternative estimators, the Jensen-Shannon divergence yields the highest overall win rate (49.8%). We also compare the quality of the estimator against the true KL divergence computed over the full distributions and discover that win rate is not necessarily correlated with low KL divergence.

KL regularization is not critical when using LoRA. While previous work (Ouyang et al., 2022; Stiennon et al., 2022) include the KL regularization penalty during PPO training, it is not necessary to achieve a high win rate within our LoRA-based experimental setup. In fact, as shown in Table 1 and Figure 1, a LoRA configuration without any KL regularization outperforms the released AlpacaFarm PPO checkpoint (48.2 vs 46.7). We conjecture that LoRA provides implicit regularization by freezing most of the parameters (e.g., feed-forward layers, layernorm, and embeddings), which already discourages large deviations from the reference policy in parameter space. Previous work on other parameter-efficient fine-tuning methods (Houlsby et al., 2019) such as prompt tuning (Lester et al., 2021; Vu et al., 2022) and prefix tuning (Li and Liang, 2021) also demonstrate less overfitting in low-data regimes, which corroborates our hypothesis. One major caveat in our experiments is that properly verifying the regularization effect of LoRA requires a comparison to full-model fine-tuning without the KL penalty, for which we do not have adequate resources.

Other divergence estimators outperform the standard KL estimator. Table 1 shows that the KL estimator used in open-source RLHF implementations such as TRLX underperforms alternative estimators. KL estimator does benefit from clamping (46.7 vs. 47.5 when clamped), but it is still worse than Jensen-Shannon regularizer (46.7 vs. 49.8) at step 100. The Jensen-Shannon estimator consistently outperforms all other estimators after step 60 in our experiments, and thus appears to be a promising alternative for future RLHF work. This finding aligns with previous work (Go et al., 2023), who also show that the Jensen-Shannon divergence outperforms other divergence measures when fine-tuning language models to approximate energy-based models.

We observe high win rates even when the KL divergence is moderately large. RLHF uses Monte-Carlo estimates of the KL divergence, and it is unclear how good these estimates are of the actual KL divergence (i.e., when computed over the entire distribution). To understand the effectiveness of minimizing the *true* KL, we sample 28K tokens from the AlpacaFarm evaluation data and plot the KL divergence in Figure 2 (left). Regardless of the estimator used, the KL divergence increases as the training proceeds. The choice of regularizer impacts the speed in which KL grows: KL divergence increases faster without any regularization ("No regularization") or when using the Bregman divergence,[16] it grows slower when regularized by the standard KL and the squared error estimators. While removing KL penalty entirely leads to larger KL (e.g., 6 times that of the standard

[16]In the Bregman divergence penalty, $\frac{\pi_{\text{ref}}(y)}{\pi_\theta(y)}$ is minimized, which encourages $\pi_\theta(y)$ to be large when $\pi_{\text{ref}(y)}$ is small despite the entropy bonus.

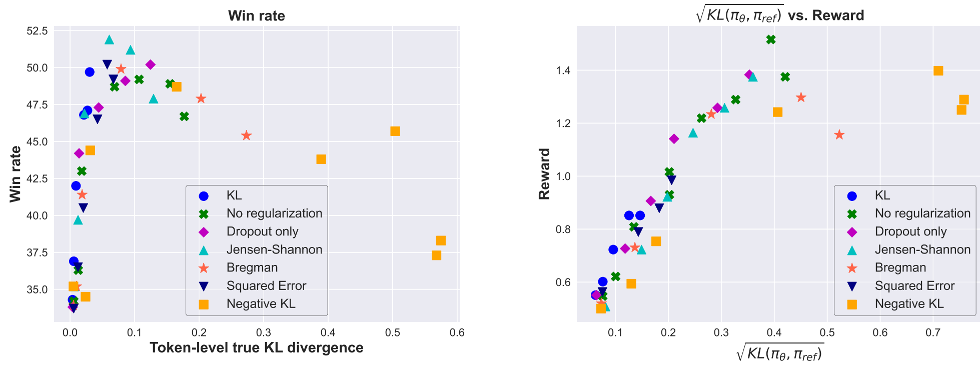

Figure 3: (**Left**): We plot the win rate vs. token-level true KL divergence for the checkpoints from steps 20, 40, 60, 80, 100, 140, and 180 of multiple configurations and multiple runs. In general win rates sharply increases and then more gradually decreases as the true KL divergence increases. (**Right**): The linear relationship between $\sqrt{\text{KL}(\pi_\theta, \pi_{\text{ref}})}$ and reward, observed in prior work, also holds in our LoRA implementation within a certain regime.

KL estimator at step 180), the resulting model still reaches win rates of $47\% \sim 49\%$, which are higher than that of the released AlpacaFarm PPO checkpoint. That said, the best win rates are achieved when the KL divergence from the reference policy is neither small nor large (on our evaluation set, around 0.05 to 0.12 per token).

A linear relationship between $\sqrt{\text{KL}(\pi_\theta, \pi_{\text{ref}})}$ and reward exists in our LoRA setup. Previous work (Bai et al., 2022a; Gao et al., 2023) demonstrates an approximately linear relationship between $\sqrt{\text{KL}(\pi_\theta, \pi_{\text{ref}})}$ and reward, which suggests that π_θ stays within a small region relative to π_{ref} (i.e., $\pi_{\text{ref}} + \delta\pi_{\text{ref}}$) during PPO training. In Figure 3 (right), we confirm that this linear relationship also holds in our LoRA setup. To go beyond the $\delta\pi_{\text{ref}}$ region, we also experiment with a configuration that *maximizes* KL during training. We find that this "negative KL" estimator leads to a region where the linear relationship breaks ($\sqrt{\text{KL}(\pi_\theta, \pi_{\text{ref}})} > 0.4$ in our experiments). In this region, reward either plateaus or starts decreasing instead of linearly increasing. This suggests that the ratio between $\sqrt{\text{KL}(\pi_\theta, \pi_{\text{ref}})}$ and the reward can be a useful metric to monitor for over-optimization during training.

We observe positive correlations between rewards, win rates, response length, and KL in certain regimes. In addition to rewards, we find that win rates and response length also positively correlate with KL within a certain regime (step < 100 and KL < 0.12 in our experiments). Going beyond this region, the average response length does not change much as KL divergence keeps increasing (Figure 2, right), whereas win rates start to drop significantly (Figure 3, left). This has practical implications on PPO training – early stopping leads to better performance while also saving compute.

PPO training has a larger negative impact on factuality with full model fine-tuning than with LoRA. To quantify the effects of PPO training on the factuality of LLM-generated text, we evaluate several checkpoints using the FActScore metric (Min et al., 2023). FActScore evaluates the factual precision of a language model by breaking a long-form response into a collection of atomic facts and then computing the precision of these atomic facts.[17] Table 2 shows that the SFT10k model (without any PPO training) obtains a higher FActScore than any checkpoint trained with PPO. Meanwhile, the released AlpacaFarm PPO checkpoint with full-model fine-tuning achieves the lowest FActScore (34.5%), with all of our LoRA-based implementations outperforming it (39.4% for the most comparable configuration). This result suggests that while PPO training can effectively steer the output to those preferred by humans for stylistic reasons (e.g., by increasing response length), it also may hurt the factuality of the generated text, and perhaps LoRA's regularization properties mitigates

[17] We use the ChatGPT+retrieval configuration of FActScore. The prompt for all of our experiments is "Tell me a bio of X", and we perform all evaluations on the labeled split of 183 people entities released by Min et al. (2023).

Configuration	FActScore (↑)	# facts per response	# tokens per response
Publicly-released AlpacaFarm checkpoints from Dubois et al. (2023)			
AlpacaFarm SFT10k	39.7%	19.3	121.1
AlpacaFarm PPO	34.5%	37.1	247.9
Our PPO models trained with LoRA			
LoRA PPO w/ KL	39.4%	26.7	170.8
LoRA PPO w/ Jensen-Shannon	38.2%	30.7	199.1
LoRA PPO w/o regularization	38.4%	33.9	217.5

Table 2: Evaluation on the FActScore labeled split (Min et al., 2023), which requires each model to generate a biography of 183 people entities. Models fine-tuned with the PPO training objective consistently underperform the SFT10k checkpoint in terms of factual precision. Implementing PPO training with LoRA somewhat mitigates the negative impact on factuality of model-generated text.

this effect to some extent. We provide example output on FActScore in Table 3 and example output on AlpacaFarm in Table 4 and Table 5 in the Appendix.

5 Related work

Our work tackles the task of aligning large language models (LLMs) to human intents via instruction-following methods (Ouyang et al., 2022). Instruction-following methods can be broadly categorized into (1) those that just perform supervised fine-tuning and (2) those that additionally apply reinforcement learning.

Methods in the first category differ in how they source instruction-following data for supervised fine-tuning. The instruction-response pairs can be generated from existing (close-sourced) models (Taori et al., 2023; Honovich et al., 2022; Mukherjee et al., 2023; Wang et al., 2023b), curated by humans (Köpf et al., 2023; Zhou et al., 2023), or even curated by LLMs (Li et al., 2023). Open-source models, when fine-tuned on high quality (Zhou et al., 2023) and diverse data (Wang et al., 2023a), can compete with blackbox LLMs on certain evaluations.

RL-based methods (Ziegler et al., 2020; Ouyang et al., 2022) align LLMs via online policy rollout and optimization. In contrast to SFT-based methods, RL incorporates feedback collected from either humans (Stiennon et al., 2022; Ouyang et al., 2022) or AI (Bai et al., 2022b; Lee et al., 2023). These judgments are distilled into a preference (reward) model for evaluating responses during online policy rollout with (sparse) rewards (Wu et al., 2023), and the whole process is more involved than the single-stage teacher-forced training used in SFT. While most RL-based methods depend on pairwise preference judgments (Dubois et al., 2023), feedback can take other forms including natural language (Saunders et al., 2022; Fernandes et al., 2023). Recently, Bansal et al. (2023) show that human feedback protocol (e.g., rating or ranking) has significant impact on the evaluation of aligned LLMs.

RL-based methods are typically regularized by a distributional term (e.g., a KL divergence penalty) to avoid degeneration caused by large deviations from a reference model. Previous work (Korbak et al., 2022b) shows that KL-regularized RL can be viewed as Bayesian inference. Minimizing KL divergence is also related to distribution matching (DM) methods (Khalifa et al., 2021), where the target optimal distribution is available. Recently, Korbak et al. (2022a) introduced KL-regularized RL from the perspective of DM, and Go et al. (2023) propose a framework that unifies KL-regularized RL, DM, and other f-divergence minimization methods.

6 Conclusion

In this technical report, we conduct an empirical analysis of the last stage of RLHF (PPO training) when implemented with low-rank adaptation, a parameter-efficient fine-tuning method. Our LoRA-based implementation of RLHF reduces the required hardware from eight to two A100 GPUs. Besides reduced memory consumption, we also observe that LoRA provides an implicit regularization effect during PPO training – good performance can be achieved even when the KL regularization term in the PPO objective is removed. Additionally, we find that alternative regularizers to the standard KL divergence estimator (e.g., Jensen-Shannon divergence) lead to higher win rates. We release our code and LoRA weights to spur future research on RLHF from under-resourced labs.

Acknowledgements

We would like to thank the AlpacaFarm authors for open-sourcing their code and models. We would like to thank Hiteshi Sharma, Banghua Zhu, Yang Liu, and Shi Dong for helpful discussions. This project was partially supported by awards IIS-1955567 and IIS-2046248 from the National Science Foundation (NSF).

References

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022b. Constitutional ai: Harmlessness from ai feedback.

Hritik Bansal, John Dang, and Aditya Grover. 2023. Peering through preferences: Unraveling feedback acquisition for aligning large language models.

Louis Castricato, Alex Havrilla, Shahbuland Matiana, Duy V. Phung, Aman Tiwari, Jonathan Tow, and Maksym Zhuravinsky. 2023. trlX: A scalable framework for RLHF.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback.

Patrick Fernandes, Aman Madaan, Emmy Liu, António Farinhas, Pedro Henrique Martins, Amanda Bertsch, José G. C. de Souza, Shuyan Zhou, Tongshuang Wu, Graham Neubig, and André F. T. Martins. 2023. Bridging the gap: A survey on integrating (human) feedback for natural language generation.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10835–10866. PMLR.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A dialogue model for academic research. Blog post.

Dongyoung Go, Tomasz Korbak, Germán Kruszewski, Jos Rozen, Nahyeon Ryu, and Marc Dymetman. 2023. Aligning language models with preferences through f-divergence minimization. *arXiv preprint arXiv:2302.08215*.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Muhammad Khalifa, Hady Elsahar, and Marc Dymetman. 2021. A distributional approach to controlled text generation. In *International Conference on Learning Representations*.

Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. 2022a. On reinforcement learning and distribution matching for fine-tuning language models with no catastrophic forgetting. In *Advances in Neural Information Processing Systems*.

Tomasz Korbak, Ethan Perez, and Christopher Buckley. 2022b. RL with KL penalties is better viewed as Bayesian inference. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1083–1091, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations – democratizing large language model alignment.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. 2023. Rlaif: Scaling reinforcement learning from human feedback with ai feedback.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-alignment with instruction backtranslation.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. *arXiv preprint arXiv:2305.14251*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. 2022. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*.

Michael Santacroce, Yadong Lu, Han Yu, Yuanzhi Li, and Yelong Shen. 2023. Efficient rlhf: Reducing the memory usage of ppo.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators.

John Schulman. 2020. Approximating kl divergence. `http://joschu.net/blog/kl-approx.html`.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2017a. Trust region policy optimization.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2018. High-dimensional continuous control using generalized advantage estimation.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017b. Proximal policy optimization algorithms.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2022. Learning to summarize from human feedback.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Tu Vu, Aditya Barua, Brian Lester, Daniel Cer, Mohit Iyyer, and Noah Constant. 2022. Overcoming catastrophic forgetting in zero-shot cross-lingual generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9279–9300, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023a. How far can camels go? exploring the state of instruction tuning on open resources.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Lilian Weng. 2018. Policy gradient. `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. 2023. Fine-grained human feedback gives better rewards for language model training.

Instruction	Tell me a bio of Don Featherstone (artist)

AlpacaFarm SFT10k Don Featherstone (1935-2015) was an American artist and sculptor best known for creating the iconic plastic pink flamingo lawn ornament. He was born in 1935 in Hartford, Connecticut and studied at the Art Students League of New York and the Yale School of Art and Architecture. He worked as an artist and sculptor for the Garden Artists Guild in New York City, creating sculptures for the Smithsonian Institution and the Metropolitan Museum of Art. In 1957, he created the plastic pink flamingo lawn ornament for the Guild, which quickly became a popular icon of the mid-20th century American suburban landscape. He continued to create sculptures and artworks throughout his life, and his work is included in the collections of the Smithsonian Institution and the Metropolitan Museum of Art. He passed away in 2015 at the age of 80.

AlpacaFarm PPO Don Featherstone (1921-1991) was an American artist and sculptor best known for creating the iconic plastic pink flamingo lawn ornament. He was born in 1921 in Los Angeles, California and studied art at the Chouinard Art Institute. He began his career as an artist and sculptor in the 1940s, creating sculptures out of clay and other materials. In the 1950s, he moved to Florida and began creating sculptures out of plaster. He was inspired to create the iconic pink flamingo lawn ornament in 1957, when he saw a flock of real pink flamingos in a Florida park. He created the first prototype out of plastic and presented it to the public the following year. The pink flamingo lawn ornament was an instant success and is now considered to be an iconic symbol of the 1950s. Don Featherstone continued to create sculptures out of plaster and other materials, but he is best known for his iconic pink flamingo lawn ornament. He passed away in 1991 in Florida.

AlpacaFarm PPO (LoRA) Don Featherstone (1935-1991) was an American artist and sculptor best known for creating the iconic plastic pink flamingo lawn ornament. He was born in 1935 in Los Angeles, California and studied art at the Chouinard Art Institute. He worked as an artist and sculptor for several companies, including Union Products and Union Plastics, before founding his own company, Don Featherstone Studios, in 1960. He is best known for creating the pink flamingo lawn ornament, which he designed in 1957 and which became a popular fixture in suburban yards in the 1960s. He also designed several other garden ornaments, including the purple swan and the blue jay. He passed away in 1991.

No regularization Don Featherstone (1935-1991) was an American artist and sculptor best known for creating the iconic plastic pink flamingo lawn ornament. He was born in 1935 in Massachusetts and studied art at the Massachusetts College of Art and the School of the Museum of Fine Arts in Boston. He began his career as an illustrator and graphic designer, creating advertisements and logos for companies such as Coca-Cola and Ford. In the late 1950s, he moved to New York City and began creating sculptures out of plaster and fiberglass. In 1957, he created his first plastic pink flamingo lawn ornament for the Union Products company. The flamingo quickly became a pop culture icon and is now considered to be a symbol of mid-century modern design. Featherstone continued to create sculptures and other works of art throughout his life, and his pieces can be found in museums and private collections around the world. He passed away in 1991 at the age of 56.

Table 3: Example of FActScore output. The actual biography of Don Featherstone can be found at `https://en.wikipedia.org/wiki/Don_Featherstone_(artist)`. Each of the responses contains hallucination to various degree. Models aligned with PPO training are less factually precise than SFT10K model.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, and Shen Li. 2023. Pytorch fsdp: Experiences on scaling fully sharded data parallel.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-tuning language models from human preferences.

Instruction Verify the correctness of the given statement. "For all integers j and k, if j and k are odd, then jk is odd."

AlpacaFarm SFT10k True.

AlpacaFarm PPO This statement is correct. For any two integers j and k, if both j and k are odd, then their product jk will also be odd. This is because the only even numbers that can be multiplied together to produce an odd number are 2 and 1. Since j and k are both odd, they can only multiply together to produce an odd number, and jk will also be odd.

AlpacaFarm PPO (LoRA) This statement is correct. For any integers j and k, if j and k are odd, then jk is also odd. This is because the product of two odd numbers is always odd. Therefore, the statement is correct.

No regularization This statement is correct. To see this, let's consider the cases where j and k are both even and where j and k are both odd. When j and k are both even, jk = (j - k) + (k - j) is even since both j - k and k - j are even numbers. When j and k are both odd, jk = (j - k) + (k - j) is odd since the sum of two odd numbers is odd. Therefore, the statement is correct.

Table 4: Example of AlpacaFarm output.

Instruction In a world where automation is becoming increasingly prevalent, is it more important to prioritize job creation or technological progress?

AlpacaFarm SFT10k It is more important to prioritize job creation in the short-term, as automation can lead to job losses and disruption of the existing economic order. In the long-term, however, technological progress should be prioritized in order to remain competitive in the global marketplace.

AlpacaFarm PPO In my opinion, it is more important to prioritize job creation in a world where automation is becoming increasingly prevalent. While technological progress is undoubtedly important, it is essential to ensure that automation does not lead to mass unemployment. Job creation should be prioritized in order to ensure that people have access to meaningful employment, as well as the income and security that comes with it. Additionally, job creation should be prioritized in order to ensure that automation does not lead to a widening of economic inequality, as automation has the potential to disproportionately benefit those who are already economically advantaged. Job creation should also be prioritized in order to ensure that automation does not lead to a decrease in human autonomy, as automation has the potential to reduce the need for human labor and therefore reduce human agency. Therefore, I believe that job creation should be prioritized in a world where automation is becoming increasingly prevalent, in order to ensure that automation does not lead to negative consequences such as mass unemployment, widening economic inequality, and a decrease in human autonomy.

AlpacaFarm PPO (LoRA) In my opinion, it is more important to prioritize job creation in the world where automation is becoming increasingly prevalent. Automation can lead to technological progress, but it can also displace workers and cause economic disruption. Therefore, it is essential to ensure that workers have access to meaningful employment and that the economy continues to grow and create new jobs. By prioritizing job creation, governments and organizations can help workers adjust to the changing job market and ensure that everyone has access to the opportunities and benefits of technological progress.

No regularization In today's world, it is more important to prioritize job creation. Automation can create efficiencies and reduce costs, but it can also displace workers and lead to job losses. Therefore, it is essential to focus on creating new job opportunities and providing people with the skills they need to succeed in the labor market. This will ensure that people have access to meaningful employment and can contribute to their communities. At the same time, it is important to encourage technological progress and innovation, as this can lead to new job creation and improved standards of living. By balancing job creation and technological progress, governments can ensure that their citizens are able to benefit from the changing economy.

Table 5: Example of AlpacaFarm output.